3/10/04

PATENT

#18

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

I hereby certify that this correspondence is being deposited with the U.S. Postal Service as first class mail in an envelope addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on March 1, 2004.

M. Ena Ellis

Appl No.	: 09/688,456	Confirmation No. 1637
Applicant	: Craig L. Ogg, et al.	
Filed	: October 16, 2000	
Title	: CRYPTOGRAPHIC MODULE FOR SECURE PROCESSING OF VALUE-BEARING ITEMS	
TC/A.U.	: 3621	
Examiner	: Firmin Backer	
Docket No.	: 39778/SAH/S850	
Customer No.	: 23363	

DECLARATION UNDER 37 CFR § 1.131

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Post Office Box 7068
Pasadena, CA 91109-7068
March 1, 2004

I, Craig L. Ogg, declare and state as follows:

1. I believe that I am a joint inventor of the subject matter described and claimed in the subject application, which claims the benefit of the following provisional patent applications: 60/160,491, filed October 20, 1999 and entitled "SECURE AND RECOVERABLE DATABASE FOR ON-LINE POSTAGE SYSTEM"; 60/160,503, filed October 20, 1999 and entitled "CRYPTOGRAPHIC MODULE ARCHITECTURE"; 60/160,112, filed October 18, 1999 and entitled "INTERNET POSTAL METERING SYSTEM"; 60/160,563, filed October 20, 1999 and entitled "SERVER ARCHITECTURE FOR ON-LINE POSTAGE SYSTEM"; 60/160,041, filed October 18, 1999 and entitled "CRYPTOGRAPHIC MODULE SECURITY APPROACH"; 60/193,057, filed March 29, 2000 and entitled "CUSTOMER GATEWAY DESIGN"; 60/193,055, filed March 29, 2000 and entitled "BROWSER-BASED IBI"; and 60/193,056, filed March 29, 2000 and entitled "MULTI-USER PSD DESIGN".

2. Before February 26, 1999, we conceived the invention claimed in this application. As employees of Stamps.com, the assignee of the invention, I assisted in the preparation of the attached specification generally describing exemplary embodiments of the invention. A true and correct copy of the specification, except for the date which has been redacted, is attached hereto as Exhibit A to this Declaration.

3. I worked diligently with our patent attorneys to prepare a series of provisional patent applications describing the subject matter set forth in the specification. Those applications were filed between October 18, 1999 and March 29, 2000 as specified hereinabove.

4. On October 16, 2000, the present patent application, claiming priority of the aforementioned provisional patent applications, was filed with the U.S. Patent and Trademark Office.

5. I declare that all statements made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under § 1001 of Title 18 of the United States Code and such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Date: 3/1/04

By: _____
Craig L. Ogg

EXHIBIT "A"

Proposed 4758 Extensions

Version 1.0

Chapter 1

Key Management

Who has what where

Key Naming Conventions

Public and Private Keys

Public keys always begin with the letter "U", private keys with "V."

Algorithm

Following U or V for asymmetric algorithms and at the beginning of symmetric algorithms will be the algorithm abbreviation. A "K" replaces the last letter to ensure that keys are misread as standing for the algorithm as a whole. Examples: DSA key = DSK, RSA key = RSK, DES key = DEK.

Purpose

The purpose of the key will be shown as a subscript. It is kept as terse as possible because of the number of times keys are mentioned.

4758 Master Key Encrypting Key

$EDEK_{4758}$ is used for all key tokens. There are other 4758 keys that may or may not need to be outlined in this document.

Indicia Signing Keys

USPS Root CA DSA Public Key ($UDSK_{IROOT}$)

$UDSK_{IROOT}$ is generated by the USPS, embedded in a certificate and is distributed according to [2]. The cryptoperiod is not specified in [2] other than to acknowledge that the key will be changed per the described changeover mechanism.

Stamps.com does not currently use this key. The infrastructure that would necessitate the use of this key has not yet been put in place.

Provider RA DSA Key Pair ($UDSK_{ISTAMPS}$, $VDSK_{ISTAMPS}$)

$UDSK_{ISTAMPS}$ and $VDSK_{ISTAMPS}$ should be generated by Stamps.com. $VDSK_{ISTAMPS}$ should be protected within the 4758's crypto-boundary. Currently, $VDSK_{ISTAMPS}$ was generated by the USPS and is stored relatively insecurely on a floppy disk.

This key pair is used to sign certificate requests for PSD DSA public keys.

PSD DSA Key Pairs ($UDSK_{PSD}$, $VDSK_{PSD}$)

For each meter a PSD DSA key pair is generated by the 4758 card. $VDSK_{PSD}$ is encrypted via the 4758 root key and stored in a database for retrieval as required and for backup. $UDSK_{PSD}$ is signed by $VDSK_{ISTAMPS}$ and sent to the USPS CA for certification.

This key pair is currently used to sign indicia and will be used to sign postage value download requests, postage value download status messages and device audit messages when the necessary USPS infrastructure is in place.

IPostage Protocol Keys

IPostage Stamps.com DSA Key Pair ($UDSK_{IPOST}$, $VDSK_{IPOST}$)

This key pair is used to verify the authenticity of the client code and during the registration state authentication protocol to sign challenges. It is generated by the 4758 and the public key portion is embedded in the host executable.

IPostage Stamps.com RSA Key-Encrypting Key Pair ($URSK_{IPOST}$, $VRSK_{IPOST}$)

This key pair is used for key exchange during the registration state. It is generated by the 4758 and the public key portion is embedded in the host executable.

IPostage HMAC Keys (HMK)

This 128-bit key, generated by the client during the registration state authentication, is used to authenticate the client during the operation state and to provide a MAC of message contents. It is stored on the host machine, DES encrypted with the PW as the key. It is changed at least every 90 days.

Meter Passphrases (PW)

Every meter will have a user chosen passphrase that is a minimum of 9 ASCII characters, contain upper and lowercase, and include at least two characters that are not letters (i.e., numbers of symbols). This gives the passphrase an approximate entropy of 56-bits. The passphrase will not be stored on the user's machine. The SHA-1 hash of the PW will be transmitted securely to the PSD and stored encrypted within the PSD package.

The user is required to change the passphrase every 6-months.

PSD Key Encrypting Triple-DES Keys ($EDEK_{PSD}$)

Every meter will have a triple-DES key that is generated by the 4758 and stored in the database in a 4758 key token. It is used to encrypt keys that were generated by the host, rather than by the 4758 (e.g., HMK and PW).

Host Passphrase-derived Key Encrypting Key ($EDEK_{PW}$)

A triple-DES key will be derived from the PW by feeding the password through SHA-1 to obtain the first block of data; incrementing the password by 1 and passing it through SHA-1 to obtain the second block of data, and then deriving the 168 bits of keying material from the first block plus 8 least significant bits from the second block.

SSL Keys

These keys actually have nothing to do with the 4758, but I wanted all of the keys listed in one place.

SSL Root CA DSA Public Key ($UDSK_{SSLCA}$)

$UDSK_{SSLCA}$ is generated by the Stamps.com's SSL CA (TBD). It is signed in the client distribution using $VDSK_{CODE}$ to preserve its integrity.

SSL PSI RSA Key Encrypting Key Pairs ($URSA_{SSL}$, $VRSA_{SSL}$)

Every server in the PSI will have an RSA key pair for SSL. The certificate for the public key will be signed using $VDSK_{SSLCA}$. Care must be taken to ensure that the domain in the certificate and the domain used to access the server are the same. Alternatively, we can use one distinguished name for all certificates and check for that specific value.

SSL Symmetric Keys

SSL symmetric keys will be generated by NSS on the client with a cryptoperiod of one SSL session. During the session, the keys are stored in general memory on both the client and the server.

Chapter 2

Basic Services

Minimal replacement layer for CCA

Introduction

CCA provides many of the basic services the current UDX layer depends on. We will need to implement enough of this to comply with FIPS level 3. Necessary services include authorization, master key services, and key tokens.

Authorization

FIPS 140-1 level 3 requires identity-based authentication.

> *Identity-Based Authentication:* A cryptographic module shall authenticate the identity of an operator and verify that the identified operator is authorized to assume a specific role (or set of roles). The module shall require that the operator be individually identified and that the specified identity be authenticated. The module shall require that the operator either implicitly or explicitly select one or more roles, and, based on the authenticated identity, verify that the operator is authorized to assume the selected roles and to request the corresponding services. The authentication of the identity of the operator, selection of roles, and verification of the authorization to assume those roles may be combined (e.g., an IC card may both identify, authenticate and authorize the operator to assume specific roles).

We plan allow an operator to authenticate and assume 1 role through that authentication. This will be implemented in a separate 4758 application that loads as the first application so that it "owns" the 4758.

Roles

The list of roles below is just a guess, I have yet to peruse the existing documentation that discusses this.

Default

Can't do anything – this is the role an operator is by default assigned to until an operator in the SysAdmin role assigns additional roles.

CryptoUser

This is the role that the server applications are assigned, it allows access to all of the extensions that do not deal with configuring the 4758.

CryptoOfficer

Not sure this role even exists, but would be the role for loading keys.

SysAdmin

Can configure the card and set the initial values of security critical parameters (e.g., transaction IDs).

Master Key

It also appears that the "master key" is a CCA concept. We need to implement a similar concept and write secret sharing functionality. We should probably do CCA one better and use a threshold approach: master key split in N parts but only N-k parts are needed to recover key.

Key Tokens

It appears that CCA was providing the key token capability we use in our API. We will have to spec and have implemented a similar architecture.

Chapter 3

Overview of 4758 Extensions

General form and rules

Introduction

Rationale

Stamps.com's PSD as represented in the 4758 card is different from traditional PSDs. It is physically distant from the host system and thus must use cryptographic techniques to ensure the authenticity and authorization of operations performed on the PSD. It serves a large number of customers and thus needs an ability to externalize its state in a secure manner.

Conceptually all the PSI does is route messages from the host to the PSD, return results generated by the PSD to the host, loads the appropriate state from the database into the PSD, and logs the audit trail provided by the PSD.

The 4758 extensions reflect the sources of data for the information it is receiving. The 4758 receives information from the host in the form of a signed *message* and externalized state from the database in a signed *package*. It indicates success or failure through a *status code* and outputs its *results* as signed data. It also outputs signed audit information as a *log entry*.

To avoid overflow and other coding errors that can lead to security holes; all functions go through a single interface that checks length, legal values, and bounds for all parameters. This interface will also validate the signature and authentication before passing control to the requested operation.

```
status_t StampOp( /*in*/ hdr4758_t* pmessage, size_t msglen,
                  /*in-out*/ package4758_t* ppackage, size_t pkglen,
                  /*out*/ hdr4758_t* presult, size_t* preslen, size_t maxreslen,
                  /*out*/ BYTE* plog, short* pnumentries, size_t* ploglen,
                         size_t maxloglen );
```

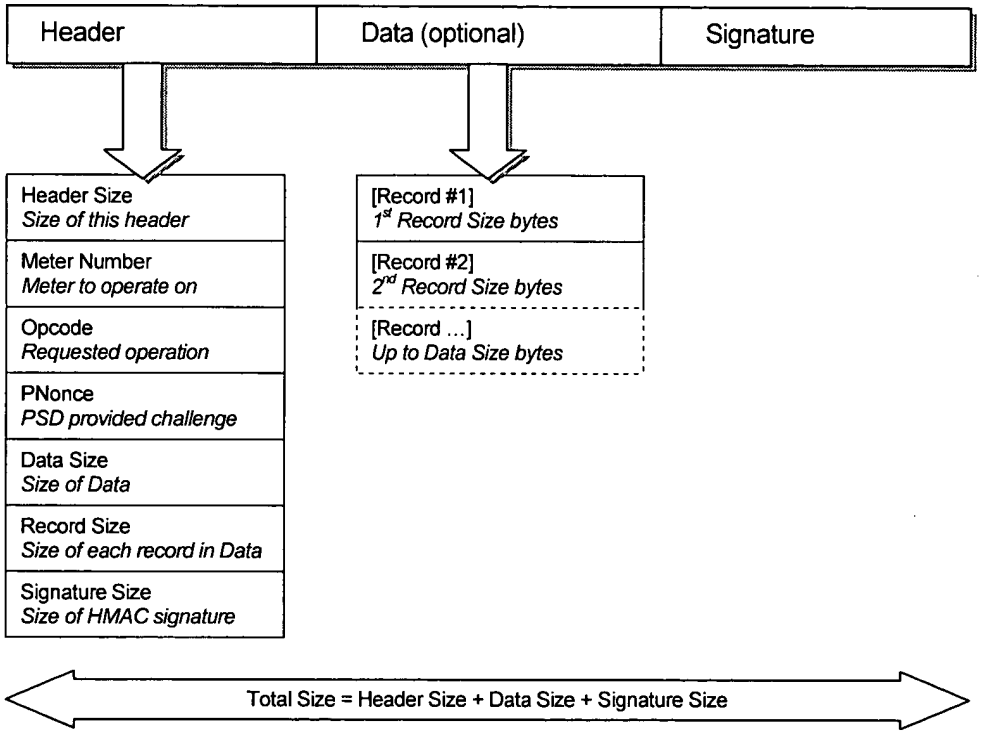
Transaction ID

There will be a global transaction ID (`transactionid`) stored in BBRAM on the 4758 that is set to 0 during the initialization of the card. This, combined with the 4758's unique ID (`cardid`), uniquely identify every transaction and allow gaps in the logs to be identified.

Message Format

A message requests a particular operation for a particular meter. Messages are composed of a header followed by data (if any) and a signature of both. It is legal for additional fields to be added to the message header without modifying the code for any individual request, but new fields must be added to the end of the header.

The signature is a 20 byte signature of the entire message generated using the HMAC-SHA1 algorithm computed according to IETF RFC 2104. The key for the HMAC will be the HMK defined above. This signature will authenticate the host, protect against malicious modification and detect transmission errors.




```
typedef DWORD id_t;
typedef WORD opcode_t;
#define CHALLENGE_SIZE 8  // unsigned 64-bit integer
#define DSAOUT_SIZE    40 // size of a DSA signature
struct hdr4758_t
{
    WORD headerSize;
    id_t meterNumber;
    opcode_t opcode;
    BYTE pnonce[CHALLENGE_SIZE];
    DWORD dataSize;
    WORD recordSize;
    WORD signatureSize;
};
```

Package Format (PSD Externalized State)

This structure includes both the data and keys, signed so that they cannot be modified while outside the crypto-boundary. Conversion of existing packages will be discussed later in the document.

```
#define INTEGRITY_SIZE 20
#define SHA_SIZE 20
#define PASSWORD_CYCLES 9
typedef unsigned long id_t;
typedef double millidollardbl_t;
typedef BYTE ciphertext_t;

struct external_keys_t {
    BYTE PW[SHA_SIZE]; // SHA-1 of passphrase
    BYTE OLDPW[PASSWORD_CYCLES][SHA_SIZE]; // make sure they don't repeat old ones

    BYTE HMK[SHA_SIZE];

    // The following optional fields are optimizations of the HMAC calculation.
    // Storing them eliminates 128 XORs every time a HMAC is calculated. Is it
    // faster to store and decrypt them than caculate them each time? Investigate.
    // We might be getting the decryption for free depending on the block size
    // of 3DES.
    BYTE HMK_INNER[SHA_SIZE];
    BYTE HMK_OUTER[SHA_SIZE];
};
```

```
struct package4758_t {
    WORD version; // Structure version (0x0001)
    MILLIDOLLARDBL_T ascReg; // The three registers
    MILLIDOLLARDBL_T descReg;
    MILLIDOLLARDBL_T postageReg;

    id_t identifier; // Account identifier

    // The following transaction ID is used to track a Postage Value Download
    // (PVD) through the entire process.  It should be incremented immediately
    // before creating a Postage Value Download Request (PVR).  Initial value
    // is 0. If there is no pending transcation, this ID corresponds to the last
    // PVD, successful or not.  If there is a pending transaction, it corresponds
    // to the pending transaction.
    id_t currentPVDTransactionID;

    // 0 if there is no pending requests, or the amount of the request if
    // there are any.  Should be cleared on a PVD or PVD error.  Initial value
    // is 0.
    MILLIDOLLARDBL_T pendingPVDAmount;
    datetime_t pendingPVDDateTime;

    // Amount and date/time of last successful PVD
    MILLIDOLLARDBL_T lastPVDAmount;
    datetime_t lastPVDDateTime;

    keytoken_t EDEK_PSD; // 3DES "external key" encrypting key
    keytoken_t DSK_PSD;  // Indicia signing DSA key pair

    // DES3 encryption = E(EDEK_PSD, external_keys_t)
    ciphertext_t externalKeys[???];

    // HMAC of this structure using the lower 512 bits of the VDSK_PSD
    BYTE integrity_value[INTEGRITY_SIZE];
};
```

The version number for this version of the register values structure is 0x0001.

Result Format

Results are formatted in the same manner as messages, with a `hdr4758_t` at the beginning. The entire result will be signed using HMAC with the HMK as the key in the same manner as messages.

Callers must take care to allocate a sufficiently large buffer or the call will fail.

Log Format

The log information that is returned may be composed of several log entries. Each log entry begins with a log header (`log4758_t`) that contains sufficient information to determine where one entry ends and the next begins. This is necessary because each

log entry must be individually signed with $VDSK_{PSD}$. An HMAC signature cannot be used because it cannot be verified by a third party without revealing the secret HMK.

```
struct log4758_t
{
    WORD headerSize;
    id_t cardid;
    id_t transactionid;
    id_t meterNumber;
    id_t certificateSerialNumber;
    opcode_t opcode;
    status_t status;
    datetime_t timestamp;  // PSD date and time in GMT timezone
    MILLIDOLLARDBL_T ascReg;
    MILLIDOLLARDBL_T descReg;
    DWORD dataSize;
    WORD signatureSize;
};
```

Chapter 4

Specifications

4758 Extensions

Registration

Sign Registration Challenge

Opcode	SDX_SIGN_REGISTRATION_CHALLENGE

Data	`struct SSRCin_t`
	`{`
	`  BYTE hnonce[CHALLENGE_SIZE];  // 64-bit challenge from host`
	`  WORD keyversion;  // Version # of key host knows`
	`};`

Package	N/A
Status	ERROR_KEY_EXPIRED, STATUS_KEY_EXPIRING, STATUS_SUCCESS

Results	`struct SSRCout_t`
	`{`
	`  BYTE hnonce[CHALLENGE_SIZE];  // 64-bit challenge from host`
	`  BYTE signature[DSAOUT_SIZE];  // S(VDSKipost, challenge)`
	`};`

Log	N/A

Note that `signaturesize` and `pnonce` are zero in the header.

Description

This opcode is used for the registration protocol in order to authenticate the PSD to the host. The `keyversion` identifies the version of $UDSK_{IPOST}$ the host expects to be used to sign the challenge. This is done to allow graceful replacement of $UDSK_{IPOST}$ when necessary.

Data

hnonce	64-bit integer challenge generated by host.
keyversion	Version number of the key the client is expecting to be used to sign the `hnonce`.

Status

STATUS_SUCCESS	Signature successfully generated.
STATUS_KEY_EXPIRING	Signature successfully generated, but key used to sign will soon expire.
ERROR_KEY_EXPIRED	Couldn't sign challenge because key requested has expired. This should never happen because other mechanisms should prevent out-of-date software from being used.

Register New User

Opcode SDX_REGISTER_NEW_USER

Data

```
struct SRNUin_t
{
   char firstname[10];
   char middleinitial;
   char lastname[40];

   char taxid[12];  // (SSN/TAXID/EID)
   char loginname[14];

   string serialid; // ??? certificate ID ???
   char email[40];
   WORD customertype;
   id_t oemid;

   // DES3 encryption = E(EDEK_PSD, external_keys_t)
   ciphertext_t externalKeys[???];
};
```

Package Out only, empty package structure should be passed in.

Status STATUS_SUCCESS, ERROR_LOGINNAME_TAKEN, ERROR_BAD_PARAMETER, ERROR_DATABASE

Results

```
struct SRNUout_t
{
   id_t customerid;
   id_t meternumber;
};
```

Log struct SRNUin_t – external keys

Note that the `pnonce` in the header will be ignored because it is not being used for authentication yet.

Description

Used to register new users.

Postage

Use Postage

Opcode	SDC_USE_POSTAGE
Record Size	
Data Size	